IN THE CLAIMS

This listing of claims will replace all prior versions, and listings, of claims in the application:

Claim 1 (Currently Amended): A system for program language processing for translating source programs to generate an object program, comprising:

a preprocessor ~~for executing~~ configured to execute preprocessing of source programs inputted in translation units;

a data type definition table, arranged for one object program, ~~for registering~~ configured to store a data type definition for data or a function in the source programs, and a use flag set in a use status when the corresponding data type definition is described in a body of any of all source programs to be linked to the one object program;

a code optimizing processor ~~for scanning~~ configured to scan all the preprocessed source programs to be used as a source for generating the object program, and ~~deleting~~ to delete, when a data type definition is already stored or the use flag is not set in a use status in the data type definition table, ~~a duplicate~~ the data type definition from the source programs to optimize the source programs;

a language processor ~~for compiling~~ configured to compile the optimized source programs; and

a software driver ~~for controlling~~ configured to control a transfer of a source program and a processing result of at least one of the preprocessor, the code optimizing processor, and the language processor.

Claim 2 (Currently Amended): The system according to claim 1, wherein the code optimizing processor includes:

a data type definition ~~detection unit for detecting~~ detector configured to detect a predetermined data type definition from the preprocessed source program;

a first decision ~~unit for deciding~~ module configured to decide whether definition information of the detected data type definition is registered into the data type definition table or not;

a first registration ~~unit for registering~~ module configured to register the definition information of the data type definition into the data type definition table when it is decided by the first decision ~~unit~~ module that the definition information of the data type definition ~~does~~ has not ~~have~~ been registered; and

a first deletion ~~unit for deleting~~ module configured to delete the data type definition detected by the data type definition ~~detection unit~~ detector from the preprocessed source program when it is decided by the first decision ~~unit~~ module that the definition information of the data type definition has been registered.

Claim 3 (Currently Amended): The system according to claim 2, wherein the code optimizing processor further includes:

an instantiation request ~~detection unit for detecting~~ detector configured to detect an instantiation request of a data type definition from the preprocessed source program;

a second decision ~~unit for deciding~~ module configured to decide, with reference to instantiation information in the data type definition table, whether the instance of a data type definition corresponding to the detected instantiation request of the data type definition has been generated or not; and

an instance ~~generation unit for generating~~ generator configured to generate the instance of the data type definition when it is decided by the second decision ~~unit~~ module that the instance of the data type definition ~~does~~ has not ~~have~~ been generated, ~~for registering~~ to

register information representing the generation of the instance into the data type definition table as the instantiation information, and

~~for suppressing generation of~~ to not generate the instance of the data type definition when it is decided by the second decision ~~unit~~ module that the instance of the data type definition has been generated.

Claim 4 (Currently Amended): The system according to claim 2, wherein the code optimizing processor further includes:

a second deletion ~~unit for deciding~~ module configured to decide the presence/absence of usage of data type in the data type definition table and ~~deleting~~ to delete a definition for a data type which is not used in all the source programs to be used as a source for generating the object program from the source programs.

Claim 5 (Currently Amended): The system according to claim 2, wherein the data type is ~~one of~~ a multiphase type data employing a template model, a multiphase type function, ~~and~~ or a multiphase type holding member function.

Claim 6 (Currently Amended): The system according to claim 5, wherein the data type definition table includes at least instantiation information representing whether instantiation of an object of a template class is requested in the source program for every ~~symbol~~ value of each data type of the multiphase type.

Claim 7 (Currently Amended): The system according to claim 6, wherein the data type definition table includes member usage information representing, when the data type is ~~a~~

the multiphase type holding member function, whether each member function is used or not, and

the optimizing processor determines member function of ~~a~~ the multiphase type the instance of which is to be actually generated in the source program with reference to the member usage information in the data type definition table.

Claim 8 (Currently Amended): The system according to claim 3, wherein the instance ~~generation unit~~ generator of the code optimizing processor converts the name of the data definition into an unique name in ~~one~~ a source program.

Claim 9 (Currently Amended): A method of program language processing for translating source programs to generate an object program, ~~comprising the steps of:~~ comprising:

executing preprocessing of source program inputted in translation units;

generating a data type definition table, arranged for one object program, for storing a data type definition for data or a function in the source programs, and a use flag set in a use status when the corresponding data type definition is described in a body of any of all source programs to be linked to the one object program;

scanning all the preprocessed source programs to be used as a source for generating the object program;

deleting, when a data type definition is already stored or the use flag is not set in a use status in the data type definition table, ~~a duplicate~~ the data type definition from the source program ~~with reference to a data type definition table arranged for one object program, the table registering a data type definition for data or a function in the source program~~ to optimize the source program; and

compiling the optimized source program.

Claim 10 (Currently Amended): The method according to claim 9, wherein the optimizing step ~~includes the steps of:~~ includes:

detecting a predetermined data type definition from the preprocessed source program;

deciding whether definition information of the detected data type definition is registered into the data type definition table or not;

registering the definition information of the data type definition into the data type definition table when it is decided that the definition information of the data type definition ~~does~~ has not ~~have~~ been registered; and

deleting the data type definition detected by the data type definition detection unit from the preprocessed source program when it is decided that the definition information of the data type definition has been registered.

Claim 11 (Currently Amended): The method according to claim 10, wherein the optimizing step further ~~includes the steps of:~~ includes:

detecting an instantiation request of a data type definition from the preprocessed source program;

deciding, with reference to instantiation information in the data type definition table, whether the instance of a data type definition corresponding to the detected instantiation request of the data type definition has been generated or not; and

generating the instance of the data type definition when it is decided that the instance of the data type definition ~~does~~ has not ~~have~~ been generated, registering information representing the generation of the instance into the data type definition table as the instantiation information, and

~~suppressing generation of~~ not generating the instance of the data type definition when it is decided that the instance of the data type definition has been generated.

Claim 12 (Currently Amended): The method according to claim 10, wherein the optimizing step further ~~includes the step of~~ includes:

deciding the presence/absence of usage of data type in the data type definition table and deleting a definition for a data type which is not used in all the source programs to be used as a source for generating the object program from the source programs.

Claim 13 (Currently Amended): The method according to claim 10, wherein the data type is ~~one of~~ a multiphase type data employing a template model, a multiphase type function, ~~and~~ or a multiphase type holding member function.

Claim 14 (Currently Amended): The method according to claim 13, wherein the data type definition table includes at least instantiation information representing whether instantiation of an object of a template class is requested in the source program for every ~~symbol~~ value of each data type of the multiphase type.

Claim 15 (Currently Amended): The method according to claim 14, wherein the data type definition table includes member usage information representing, when the data type is a multiphase type holding member function, whether each member function is used or not, and

the optimizing step determines member functions of ~~a~~ the multiphase type the instance of which must be actually generated in the source program with reference to the member usage information in the data type definition table.

Claim 16 (Currently Amended): A computer readable recording medium for causing a computer to execute program language processing for translating a source program to generate an object program, comprising:

a process for executing preprocessing of source program input in translation units;

a process for generating a data type definition table, arranged for one object program, for storing a data type definition for data or a function in the source programs, and a use flag set in a use status when the corresponding data type definition is described in a body of any of all source programs to be linked to the one object program;

a process for scanning all the preprocessed source programs to be used as a source for generating the object program;

a process for deleting, when a data type definition is already stored or the use flag is not set in a use status in the data type definition table, ~~a duplicate~~ the data type definition from the source programs with reference to a data type definition table arranged for one object program so as to suppress instantiation of a data type definition which has been instantiated as needed to optimize the source ~~program, the table registering a data type definition for data or a function in the source~~ program; and

a process for compiling the optimized source program.

Claim 17 (Currently Amended): The medium according to claim 16, wherein the optimizing process includes:

a process for detecting a predetermined data type definition from the preprocessed source programs;

a process for deciding whether definition information of the detected data type definition is registered into the data type definition table or not;

a process for registering the definition information of the data type definition into the data type definition table when it is decided that the definition information of the data type definition ~~does~~ has not ~~have~~ been registered; and

a process for deleting the data type definition detected by the data type definition detection process from the preprocessed source program when it is decided that the definition information of the data type definition has been registered.

Claim 18 (Currently Amended): The medium according to claim 17, wherein the optimizing process further includes:

a process for detecting an instantiation request of a data type definition from the preprocessed source program;

a process for deciding, with reference to instantiation information in the data type definition table, whether the instance of a data type definition corresponding to the detected instantiation request of the data type definition has been generated or not; and

a process for generating the instance of the data type definition when it is decided that the instance of the data type definition ~~does~~ has not ~~have~~ been generated, registering information representing the generation of the instance into the data type definition table as the instantiation information, and

~~suppressing generation of~~ not generating the instance of the data type definition when it is decided that the instance of the data type definition has been generated.

Claim 19 (Currently Amended): A program product for causing a computer to execute program language processing for translating source programs to generate an object program, comprising:

a process for executing preprocessing of source program input in translation units;

a process for generating a data type definition table, arranged for one object program, for storing a data type definition for data or a function in the source programs, and a use flag set in a use status when the corresponding data type definition is described in a body of any of all source programs to be linked to the one object program;

a process for scanning all the preprocessed source programs to be used as a source for generating the object program;

a process for deleting, when a data type definition is already stored or the use flag is not set in a use status in the data type definition table, ~~a duplicate~~ the data type definition from the source program ~~with reference to a data type definition table arranged for one object program so as to suppress instantiation of a data type definition which has been instantiated as needed~~ to optimize the source ~~program, the table registering a data type definition for data or a function in the source~~ program; and

a process for compiling the optimized source program in units of translation.

Claim 20 (Currently Amended): The program product according to claim 19, wherein the optimizing process includes:

a process for detecting a predetermined data type definition from the preprocessed source program;

a process for deciding whether definition information of the detected data type definition is registered into the data type definition table or not;

a process for registering the definition information of the data type definition into the data type definition table when it is decided that the definition information of the data type definition ~~does~~ has not ~~have~~ been registered; and

A16

a process for deleting the data type definition detected by the data type definition detection process from the preprocessed source program when it is decided that the definition information of the data type definition has been registered.

Claim 21 (New): A system for program language processing for translating source programs to generate an object program, comprising:

a preprocessor for executing preprocessing of source programs inputted in translation units;

a multiphase data type definition table, arranged for one object program, for storing a multiphase data type definition employing a template model for data or a function in the source programs, and a use flag set in a use status when the corresponding multiphase data type definition is described in a body of any of all source programs to be linked to the one object program;

A17

a first code optimizing processor for scanning all the preprocessed source programs to be used as a source for generating the object program, and deleting, when a multiphase data type definition is already stored or the use flag is not set in a use status in the multiphase data type definition table, the multiphase data type definition from the source programs to optimize the source programs;

a second code optimizing processor for scanning all the preprocessed source programs, and generating instance of the multiphase data type description in the body of the source programs only when the corresponding use flag is not set in a use status in the multiphase data type definition table;

a language processor for compiling the optimized source programs; and

a software driver for controlling a transfer of a source program and a processing result of at least one of the preprocessor, the code optimizing processor, and the language processor.

Claim 22 (New): A method of program language processing for translating source programs to generate an object program, comprising:

executing preprocessing of source programs inputted in translation units;

generating a multiphase data type definition table, arranged for one object program, for storing a multiphase data type definition employing a template model for data or a function in the source programs, and a use flag set in a use status when the corresponding multiphase data type definition is described in a body of any of all source programs to be linked to the one object program;

scanning all the preprocessed source programs to be used as a source for generating the object program;

deleting, when a multiphase data type definition is already stored or the use flag is not set in a use status in the multiphase data type definition table, the multiphase data type definition from the source programs to optimize the source programs;

generating instance of the multiphase data type description in the body of the source programs only when the corresponding use flag is not set in a use status in the multiphase data type definition table; and

compiling the optimized source programs.

Claim 23 (New): A computer readable recording medium for causing a computer to execute program language processing for translating source program to generate an object program, comprising:

a process for executing preprocessing of source programs inputted in translation units;

a process for generating a multiphase data type definition table, arranged for one object program, for storing a multiphase data type definition employing a template model for data or a function in the source programs, and a use flag set in a use status when the corresponding multiphase data type definition is described in a body of any of all source programs to be linked to the one object program;

a process for scanning all the preprocessed source programs to be used as a source for generating the object program;

a process for deleting, when a multiphase data type definition is already stored or the use flag is not set in a use status in the multiphase data type definition table, the multiphase data type definition from the source programs to optimize the source programs;

A7

a process for generating instance of the multiphase data type description in the body of the source programs only when the corresponding use flag is not set in a use status in the multiphase data type definition table; and

cl B1

a process for compiling the optimized source programs.

Claim 24 (New): A program product for causing a computer to execute program language processing for translating source programs to generate an object program, comprising:

a process for executing preprocessing of source programs inputted in translation units;

a process for generating a multiphase data type definition table, arranged for one object program, for storing a multiphase data type definition employing a template model for data or a function in the source programs, and a use flag set in a use status when the corresponding multiphase data type definition is described in a body of any of all source programs to be linked to the one object program;

a process for scanning all the preprocessed source programs to be used as a source for generating the object program;

a process for deleting, when a multiphase data type definition is already stored or the use flag is not set in a use status in the multiphase data type definition table, the multiphase data type definition from the source programs to optimize the source programs;

a process for generating instance of the multiphase data type description in the body of the source programs only when the corresponding use flag is not set in a use status in the multiphase data type definition table; and

a process for compiling the optimized source programs.